ECE385

Fall 2021

Experiment 8

SOC with USB and VGA Interface in SystemVerlog

Ge Yuhao, Lou Haina
D231, Nov.25, 2021
TA: Chenhao Wang

1 Introduction

In this lab we implement a USB and VGA system which can control the ball's direction on a VGA monitor according to the USB keyboard inputs. The keys WASD are used to control the direction of the ball, and the ball will bounce if it reaches the boundary of the screen.

2 Written Description of Lab 8 System

2.1 Written Description of the entire Lab 8 system

- In Lab8 we have two more connection to the DE2 board. One is a keyboard connected with a USB port, the other is a monitor connected with a VGA line.

- We still use Nios II embedded processor in this Lab. The USB protocol is handled in the software on the Nios II, and the extracted keycode from the USB keyboard is then sent to the hardware. With the keyboard input, the balls movement can be determined through some SystemVerlog logic to realise the direction changing and bouncing.

- We use a simple color mapper combined with the VGA controller to draw simple shapes on the monitor. , We only need to draw the ball's location and the background image at every frame and set the screen refresh rate (60 Hz), the monitor will show the movement of the ball continuously.

- When the program starts, a stationary red ball should be displayed in the center of the screen. The ball should be waiting for a direction signal from the keyboard. As soon as a direction key (W-A-S-D) is pressed on the keyboard, the ball will start moving in the direction specified by the key. When the ball reaches the edge of the screen, it should bounce back and start moving in the opposite direction. The ball will keep moving and bouncing until another command is received from the keyboard. When a different direction key is pressed, the ball should start moving in the newly specified direction immediately, without returning to the center of the screen.

2.2 Describe in words how the NIOS interacts with both the CY7C67200 USB chip and the VGA components

USB The USB port on the DE2 board is equipped with the Cypress EZ-OTG (CY7C67200) USB Controller, which handles all the data transmission via the physical USB port and manages structured information to and from the DE2 board. In this Lab we use CY7C67200 as a Host Controller to make the FPGA accept the input from the keyboard. First we write a hardware I/O wrapper to tri-state the NIOS II driven data bus and to handle setting the CY7C67200 control pins. Then we write two functions: IO_read and IO_write, to write to one of the four registers in the register bank (HPI_ADDRESS, HPI_DATA, HPI_MAILBOX, or HPI_STATUS) on the CY7C67200 chip. Next we write USBRead and USBWrite to define the proper sequence of writes and reads to use those registers to access the CY7C67200's memory.

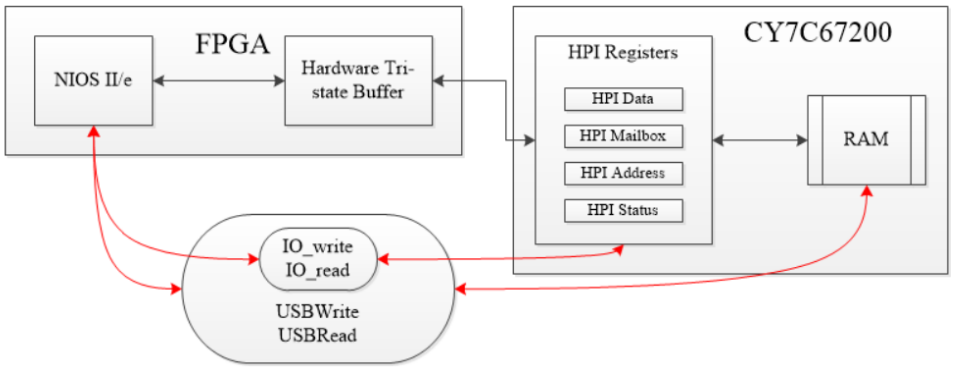

Figure 1: Connection between NIOS and CY7C67200

VGA We have VGA_controller and Color_Mapper modules to handle the display on the monitor. The VGA_controller module produces the timing signals to ensure proper synchronization with the DE2's VGA output. The Color_Mapper module is used to handle object rendering and coloring, which draw the picture for each frame. When we use the 50 MHz clock of the FPGA, we can use a 60Hz frame rate for the VGA output.

2.3 Written description of the USB protocol

We write four functions USBRead, USBWrite, IO_read and IO_write in C code to enable reading and writing through USB protocol.

USBRead Reads data from the registers in the CY7C67200 USB Controller. Instantiates one use of IO_Write for the address and returns the data found in that address via IO_Read.

USBWrite Instantiates two uses of IO_Write per function in order to write the address and data values to the internal registers of the USB controller.

IO_Read Returns the 16 bit data held in otg_hpi_data after updating otg_hpi_address to a given 8 bit input address.

IO_Write Writes data from the Address and Data inputs to the address and data pointers to the USB Controller registers.

2.4 Block diagram

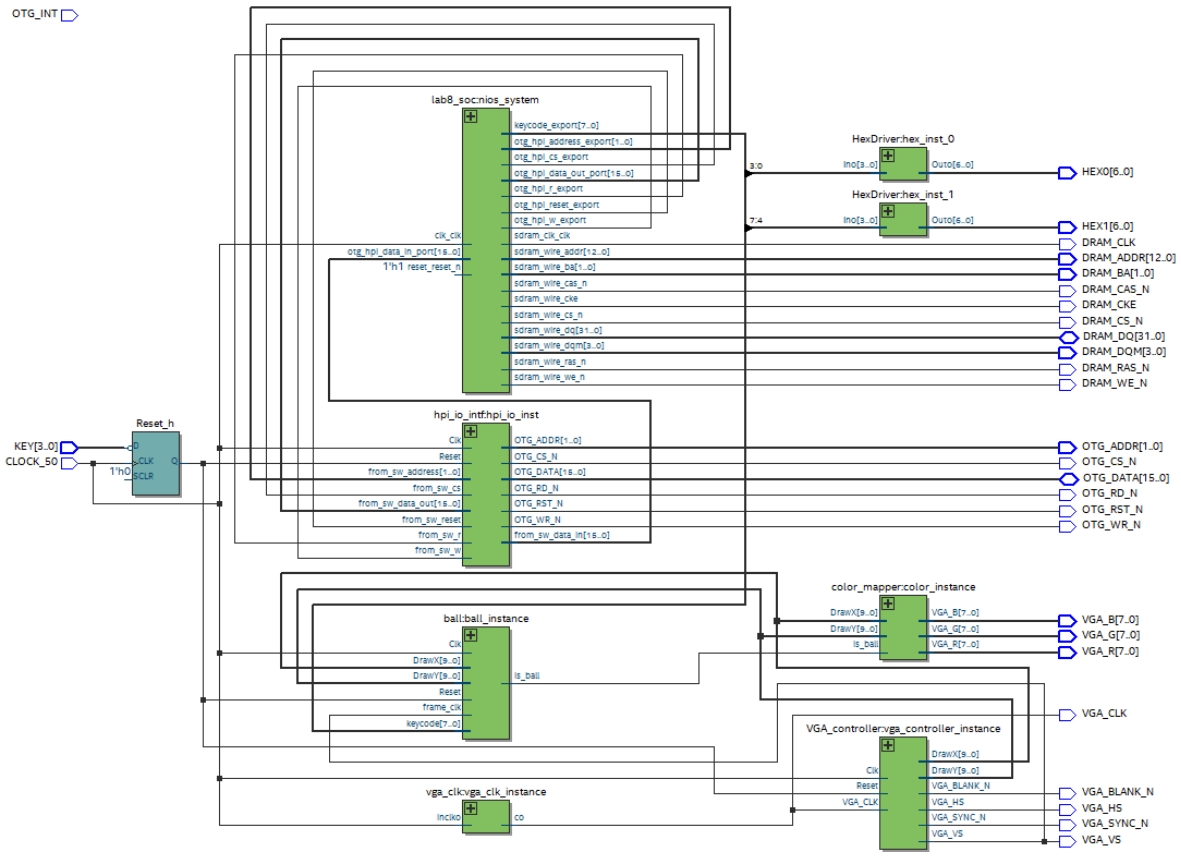

Figure 2: Top level Block Diagram

2.5 Module descriptions

2.5.1 Quartus modules

Module: Ball
Inputs: Clk, Reset, frame_clk, keycode, DrawX, DrawY
Output: is_ball
Description: This module creates the ball instance and use SystemVerlog logic to control the moving and boucing of the ball.
Purpose: It makes the ball be able to move in x+, x-, y+, y- depending on the keyboard inputs, and make it to bounce back when reaching the wall.

Module: color_mapper
Inputs: DrawX, DrawY, is_ball
Output: VGA_R, VGA_G, VGA_B
Description: Decide which color to be output to VGA for each pixel
Purpose: This module can draw the picture of each frame according to the ball's position.

Module: HexDriver
Inputs: Input
Output: Output
Description: The ASCII character value will be displayed onto the HexDriver of the DE2 board/

Purpose: The keyboard input will be displayed onto the HexDrive of the DE2 board, which helps us to know if the keyboard functions well.

Module: Lab8
Inputs: CLOCK_50, KEY, OTG_INT
Output: HEX0, HEX1, VGA_R, VGA_G, VGA_B, VGA_CLK, VGA_SYN_N, VGA_BLANK_N, VGA_VS, VGA_HS, OTG_DATA, OTG_ADDR, OTG_CS_N, OTG_RD_N, OTG_WR_N, OTG_RST_N, OTG_INT,DRAM_ADDR, DRAM_DQ, DRAM_BA, DRAM_DQM, DRAM_RAS_N, DRAM_CAS_N, DRAM_CKE, DRAM_WE_N, DRAM_CS_N, DRAM_CLK
Description: This is the top level module of lab 8.
Purpose: This module connect all the other modules, and control the port communication of different modules.

Module: VGA_controller
Inputs: Clk, Reset, VGA_CLK
Output: VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N, DrawX, DrawY
Description: Takes the data of the ball's current position on screen and updates it to the VGA display. Use the vertical and horizontal syncs signal to control the drawing of the VGA display.
Purpose: This module set the configuration of the screen such as the size, edge, and sync signals. It also records the present pixel we are drawing which enables the displaying function.

Module: hpi_io_intf
Inputs: Clk, Reset, from_sw_address, from_sw_data_out, from_sw_r, from_sw_w, from_sw_cs, from_sw_reset
Output: from_sw_data_in, OTG_ADDR, OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N
inout: OTG_DATA
Description: This module is the interface between NIOS II and EZ-OTG chip.
Purpose: This module sends the read, write, address, databuffer signals to the OTG chip to ensure correct data reads and writes. It also allows us to control the input output bus.

Module: vga_clk
Inputs: inclk0
Output: c0
Description: This module produce the clock signal which is used to control the VGA_controller.
Purpose: This module generates the 25MHz pixel clock for the VGA controller which allows the VGA controller to update the screen at clock to 60Hz.

2.5.2 Qsys Modules

nios2_gen2_0: The nios2 block helps us to translate the C code into the SystemVerilog which can be excuted on the FPGA board.

Onchip_memory2_0 This is the memory block for the CPU which can store some data for the proigram.

Clk_0 Serves as the general clock for the entire system, which control all the other modules' clk signal. It can also reset the clk signal for other modules.

sdram As the space of Onchip_memory is limited, this block add another SDRAM to our system. And we have to use an SDRAM controller to interface.

sdram_pll This module generates the clock for the SDRAM. The pll add delay (3ns) to the SDRAM with delays to wait for the outputs to stabilize.

sysid_qsys_0 This block verifies the correct transfer of the software and hardware by looking back and forth between the C code and SystemVerilog to assure the data transfer is done in the correct format.

keycode The inputs are the buttons on the FPGA device. This PIO allows for data transfer between the hardware and the software.

jtag_uart_0 Allows for terminal access for use in debugging the software. keycode: Reads data, updates the USB chip and then sends it to the software for logic instruction.

otg_hpi_address PIO that allows the desired address in memory of the SoC to be found that is sent from the software to the FPGA.

otg_hpi_data PIO that allows for cross transfer of data from the FPGA to software and the other way around. This PIO has a width of 16 bits which allows for sizable data transfer between the two.

otg_hpi_r PIO that allows the enable signal to read from the memory of the SoC.

otg_hpi_w PIO that allows the enable signal to write to the memory of the SoC.

otg_hpi_cs PIO that allows the enable signal turn on and off the memory of the SoC.

otg_hpi_reset PIO that allows for the reset signal of the memory of the SoC.

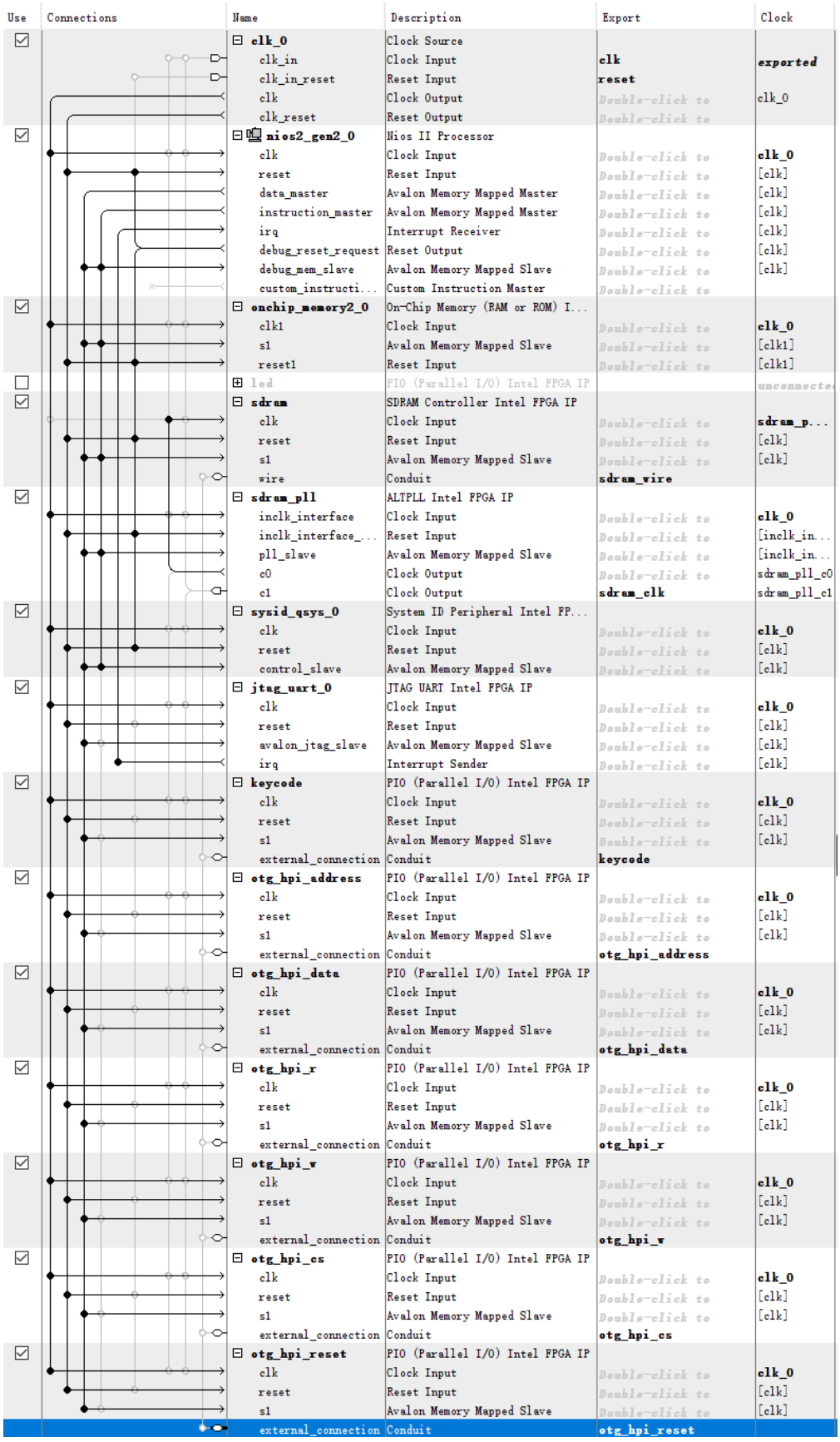

Figure 3: View of the platform designer

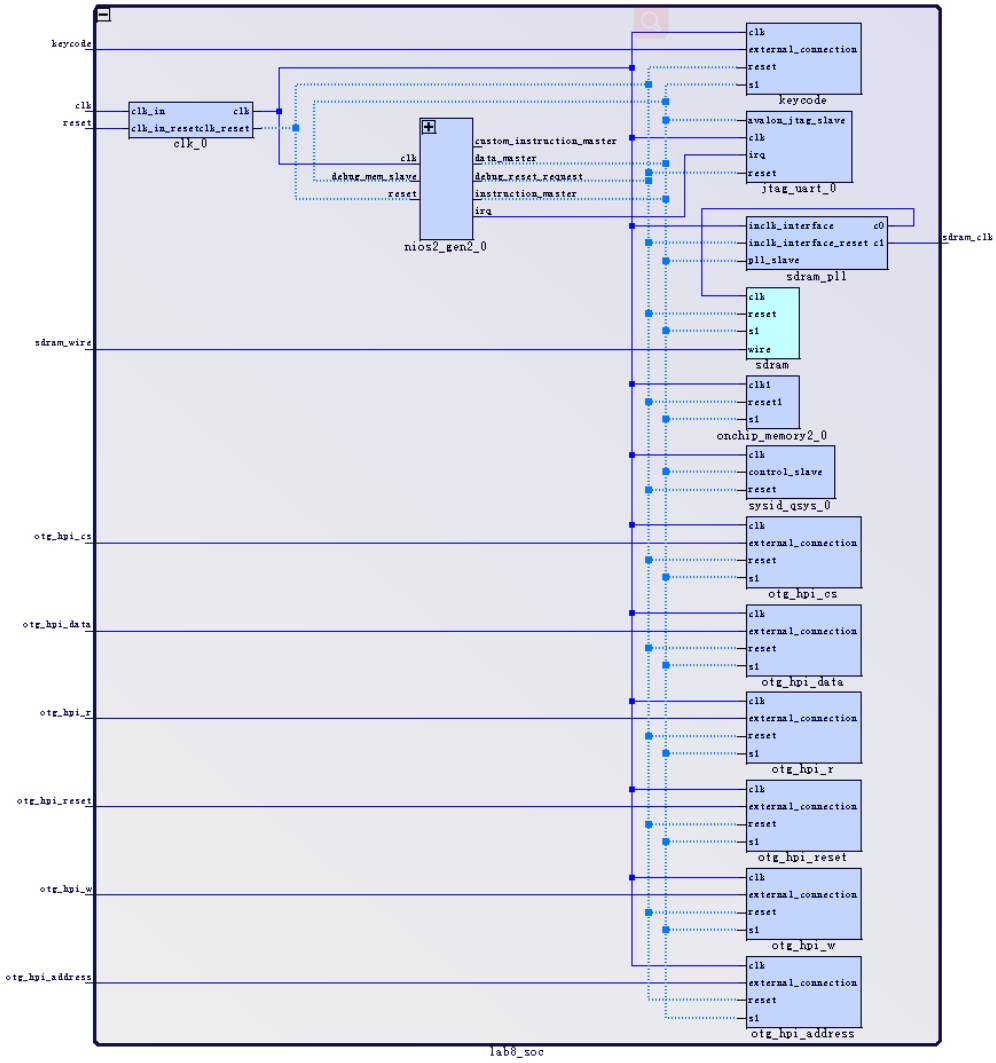

Figure 4: schematic of the platform designer

3 Answers to both hidden questions

3.1 What are the advantages and/or disadvantages of using a USB interface over PS/2 interface to connect to the keyboard? List any two.

- USB interface supports hot swap, and external device can be directly used immediately after plugging in the usb interface, while we need to restart computer to use external device through PS/2 interface. It is more convenient.

- The scanning frequency of USB ports is much higher than that of PS/2 ports.

- For keystroke combination, we can only press 6 keys at the same time through USB interface, because only 8 bytes data can be transferred in one data package. Therefore, USB keyboard can only record 6 states of keystrokes at the same time. And no conflicts through PS/2 interface.

3.2 Notice that Ball_Y_Pos is updated using Ball_Y_Motion. Will the new value of Ball_Y_Motion be used when Ball_Y_Pos is updated, or the old? What is the difference between writing "Ball_Y_Pos_in = Ball_Y_Pos + Ball_Y_Motion;" and "Ball_Y_Pos_in = Ball_Y_Pos + Ball_Y_Motion_in;"? How will this impact behavior of the ball during a bounce, and how might that interact with a response to a keypress?

- When Ball_Y_Pos is being updated, the old value of Ball_Y_Motion will be used due to parallel assignments.
- If we use Ball_Y_Motion, the Y position of the current clock cycle is determined by old Y position and old motion in the Y direction, which will not be influenced by the old pressed key or whether the ball is at the boundary.
- If we use Ball_Y_Motion_in, the Y position of the present clock cycle is determined by old Y position, old key-press action and the boundary status.
- Using Ball_Y_Motion_in, the reaction of the ball will be faster. When the ball reaches the wall, it will go back immediately, and when the keypress happens, the direction will change immediately. Using Ball_Y_Motion won't realize this as the reaction of the picture is one frame slower than the code.

4 Postlab Question

4.1 What is the difference between VGA_clk and Clk?

The VGA_clk is approximately 25 MHZ for pixel frequency to update frames on the VGA monitor, while clk is 50 MHZ to drive FPGA and other devices.

4.2 In the file io_handler.h, why is it that the otg_hpi_data is defined as an integer pointer while the otg_hpi_r is defined as a char pointer?

An integer is 32 bits and a char is 8 characters. As the data is 32 bits wide, we need otg_hpi_data to be an integer pointer which point to inout data. But for otg_hpi_r, we only use it to point to a register which may contain 0 or 1 so 1 byte is enough for this. By doing this, we can save memory as we only use the necessary part of the memory.

4.3 Document the Design Resources and Statistics in following table.

LUT	2694
DSP	10
Memory(BRAM)	11392 bits
Flip-Flop	2219
Frequency	133.37MHz
Static Power	105.16mW
Dynamic Power	0.79mW
Total Power	177.82mW

Table 1: Design Resources and Statistics

5 Conclusion

5.1 Discuss functionality of your design. If parts of your design didn't work, discuss what could be done to fix it

Our design works perfectly. We have done most of our work smoothly but we still meet some bugs. For example, when we finish our design, we found that if we press W and D button when the ball is at boundary, it will move diagonally or leave the monitor at the corner. After checking our code in ball.sv, we find the ball meet two requirements of updating the direction of velocity, therefore it will have both x and y velocity. We put press key condition in 'else' in order to make either bounce or change direction by pressing key, and then it functions correctly and never bounce out of boundary.

5.2 Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab manual or given materials which can be improved for next semester?

Most of part in lab manual is well. I think if you could add some overall block diagram of the whole structure and how software and hardware are related, it will be easier for us to understand. And it is better to include the connection of interface file with eclipse, platform designer and other devices.